5. Feature Location and Traceability

(Last Update: 2024-03-06)

Lots of research on software engineering has been been devoted on finding solutions which make maintenance of the software system easier. Software development is based on multiple iterations where the code is incremented, modified and pruned. During this process, the developer receives the task and then starts looking for the part of the code that has to be updated. Unfortunately, there is mismatch between the words used in the task and the code that has to be modified. The task is described in natural language whereas the code is written in a programming language. As a consequence, before the developer identifies the right section of code, he or she has to skim among software modules (e.g. files).

Feature location is the activity of identification of the source code related to software functionalities. A feature, generally described by using natural language, defines a functionality requirement of the system. Feature location is a fundamental activity for impact analysis [2] and incremental change process [3].

Traceability of requirements has been described as "the ability to follow the life of a requirement, in both a backward and forward direction" [1]. Traceability is important when the software has several years of development, many features, and lots of developers.

The interested reader may refer to the following to discover the new trends of the field and for more information on feature location and traceability.

- Dit, B., Revelle, M., Gethers, M. and Poshyvanyk, D. (2013), Feature location in source code: a taxonomy and survey. J. Softw. Evol. and Proc., 25: 53-95. https://doi.org/10.1002/smr.567 (https://onlinelibrary.wiley.com/doi/full/10.1002/smr.567)
- Batot, E.R., Gérard, S., Cabot, J. (2022). A Survey-driven Feature Model for Software Traceability Approaches. In: Johnsen, E.B., Wimmer, M. (eds) Fundamental Approaches to Software Engineering. FASE 2022. Lecture Notes in Computer Science, vol 13241. Springer, Cham. https://doi.org/10.1007/978-3-030-99429-7_2 (https://link.springer.com/chapter/10.1007/978-3-030-99429-7_2)

In this session we will analyze Jedit (download the workspace).

First explore the MyJ project manually. The project contains the source for a text editor. Imagine you have to add a feature where the color of the caret needs to be changed when it's blinking. It would be difficult to know how to get started when you don't know the project structure.

Install Conqueror in Eclipse by following the install instructions.

You can use conqueror by clicking on the "search" icon in the top bar and selecting "Conquer search". Use Conqueror to search for "caret blinking".

- Which methods does conqueror find that are relevant?
- Which alternatives used by conqueror are usefull? Which are not?

Perform another conqueror search using the terms "change caret color".

- Which methods does conqueror find that are relevant?
- Which alternatives used by conqueror are usefull? Which are not?
- With the two searches combined, do you know have a better understanding of how you would implement the new feature?

[1] Olly Gotel, Anthony Finkelstein: Contribution structures (Requirements artifacts). RE 1995: 100-107.
[2] R. S. Arnold, Software Change Impact Analysis. IEEE Computer Society Press, 1996.
[3] V. Rajlich and P. Gosavi, "Incremental Change in Object-Oriented Programming," IEEE Software, vol. 21, no. 4, pp. 62–69, 2004.

*If you have any trouble configuring the tools, you may download a ready to use version of Eclipse Juno.